Test Data Compression Using Dictionaries with Fixed-Length Indices*

Lei Li and Krishnendu Chakrabarty
Department of Electrical and Computer Engineering
Duke University, Durham, NC 27708
{ll, krish}@ee.duke.edu

***Abstract*—We present a dictionary-based test data compression approach for reducing test data volume and testing time in SOCs. The proposed method is based on the use of a small number of ATE channels to deliver compressed test patterns from the tester to the chip and to drive a large number of internal scan chains in the circuit under test. Therefore, it is especially suitable for a reduced pin-count and low-cost DFT test environment, where a narrow interface between the tester and the SOC is desirable. The dictionary-based approach not only reduces testing time but it also eliminates the need for additional synchronization and handshaking between the SOC and the ATE. The dictionary entries are determined during the compression procedure by solving a variant of the well-known clique partitioning problem from graph theory. Experimental results for the ISCAS-89 benchmarks and representative test data from IBM show that the proposed method outperforms a number of recently-proposed test data compression techniques.**

I. Introduction

Intellectual property (IP) cores are now being routinely used in large system-on-a-chip (SOC) designs. Higher circuit densities and a larger number of embedded cores lead to higher test data volume, which in turn leads to an increase in testing time. New techniques are therefore needed to reduce test data volume and testing time, as well as to overcome ATE memory and bandwidth limitation. In order to facilitate test reuse and the use of IP cores, these techniques should not require detailed structural models for additional fault simulation or test generation.

Built-in self-test (BIST) offers a promising alternative to ATE-based external testing. While BIST is now extensively used for memory testing, it is less common for logic testing. Problems with logic BIST include inadequate fault coverage due to random-resistant fault and bus contention during test application. While the fault coverage can be made quite high using methods such as reseeding [11], bit-flipping [26] and bit-fixing [22], these techniques require structural information for fault simulation and test generation.

Structural methods for reducing test data volume and testing time require design modifications. For example, the Illinois scan architecture (ILS) offers an alternative to conventional scan design [12]. However, fault simulation and test generation are necessary in ILS as post-processing steps to get high fault coverage.

Test data compression is a non-intrusive method that can be used to compress the precomputed test set T_D provided by the core vendor to a much smaller test set T_E, which is then stored in ATE memory. An on-chip decoder is used to generate T_D from T_E during pattern application. A number of techniques based on statistical coding [13, 15], run-length coding [14], Golomb coding [4], FDR coding [3], EFDR coding [7], and VIHC coding [9], have been proposed to reduce test data volume. Test data volume reduction techniques based on on-chip pattern decompression are also presented in [2, 6, 18, 21, 23]. Several dictionary-based compression methods have recently been presented to reduce SOC test data volume. In [15], frequently occurring blocks are encoded into variable-length indices using Huffman coding. A dictionary with fixed-length indices is used to generate all the distinct output vectors in [19]. A test data compression technique based on LZ77 algorithm, which uses a dynamic dictionary, is proposed in [25].

The resurgence of interest in test data compression has also led to new commercial tools that can provide over 10X compression for large industrial designs. For example, the OPMISR [1] and SmartBIST [16] tools from IBM and the TestKompress tool from Mentor Graphics [17] reduce test data volume and testing time through the use of test data compression and on-chip decompression.

In this paper, we present a new dictionary-based test data compression method that provides significant compression for precomputed test sets and leads to considerable reduction in testing time. The dictionary uses fixed-length indices, and its entries are carefully selected such that the dictionary is efficiently utilized. The proposed method is based on the use of a small number of ATE channel to drive a large number of internal scan chains in the core under test; see Figure 1. (The test response can be compacted using a MISR or other techniques.) Unlike coding techniques such as [3, 4], this approach does not require multiple clock cycles to determine the decompressed test pattern after the last bit of the corresponding compressed data packet is transferred from the ATE to the chip. This dictionary-based approach therefore not only reduces testing time but it also eliminates the need for additional synchronization and handshaking between the SOC and the ATE. This approach is therefore targeted towards a reduced pin-count test and low-cost DFT tester [24] environment, where a narrow interface between the tester and the SOC is desirable.

The rest of the paper is organized as follows. In Section II, we briefly introduce dictionary-based data compression and show how a dictionary can be used for test data compression. We describe how a variant of the clique partitioning problem from graph theory can be used for the compression procedure. Section III describes the decompression architecture. Exper-

*This research was supported in part by the National Science Foundation under grants CCR-9875324 and CCR-0204077, and by a graduate fellowship from the Design Automation Conference.

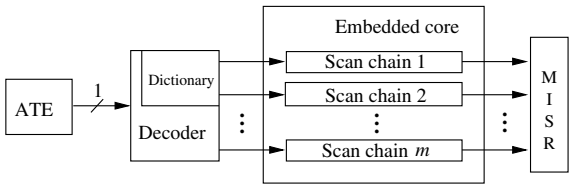

Fig. 1. Illustration of the proposed method for a single ATE channel.

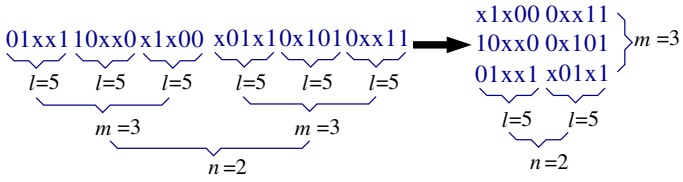

Fig. 2. An example of formatting the test data for multiple scan chains.

imental results and a comparison with related recent work are presented in Section IV. Finally, Section V concludes the paper.

II. Dictionary-Based Test Data Compression

Dictionary-based methods are quite common in the data compression domain [20]. While statistical methods use a statistical model of the data and encode the symbols using variable-size codewords according to their frequencies of occurrence, dictionary-based methods select strings of the symbols to establish a dictionary, and then encode them into equal-size tokens using the dictionary. The dictionary stores the strings, and it may be either static or dynamic (adaptive). The former is permanent, sometimes allowing for the addition of strings but no deletions, whereas the latter holds strings previously found in the input stream, allowing for additions and deletions of strings as new input is processed.

The well-known compression algorithm LZ77 [20] is based on a dynamic dictionary. It uses part of the previously-seen input stream as the dictionary. Since our proposed method uses only a static dictionary, details of a dynamic dictionary are not discussed here.

We next describe the proposed dictionary-based test data compression method and illustrate it with an example. In the following description, we assume that the precomputed SOC test data T_D consists of n test patterns $t_1, t_2, \ldots, t_n$. The scan elements of the core under test are divided into m scan chains in as balanced a manner as possible. Each test vector can therefore be viewed as m subvectors. If one or more subvectors are shorter than the others, don't-cares are padded to the end of these subvectors so that all the subvectors have the same length, which is denoted by l. The m-bit data at the same position of each subvector constitute an m-bit *word*. A total of nl m-bit words thus are formed and encoded during the compression procedure. Figure 2 illustrates the formatting of the given test data for multiple scan chains. During test application, after a codeword is shifted into the decoder, an m-bit word $u_1, u_2, \ldots, u_m$ is immediately generated by the decoder and fed into the scan chains (one bit for each scan chain).

In the dictionary-based test data compression method, each codeword is composed of a prefix and a stem. The prefix is a 1-bit identifier that indicates whether the stem is a dictionary index or a word of uncompressed test data. If it equals 1, the stem is viewed as a dictionary index. On the other hand, if the prefix equals 0, the stem is an uncompressed word and it is m bits long. The length of the dictionary index depends on the size of the dictionary. If D is the set of the entries in the dictionary, the length of the index $l_{index} = \lceil \log_2 |D| \rceil$, where $|D|$ is the size of the dictionary. Since l_{index} is much smaller than m, the compression efficiency is greater if more test data words can be obtained from the dictionary. However, the dictionary must be reasonably small to keep the hardware overhead low. Fortunately, since there are many don't-care bits in scan test data for typical circuits, we can appropriately map these don't-care bits to binary values and carefully select the entries for the dictionary, so that as many words as possible are mapped to the entries in the dictionary.

An important step in the compression procedure is that of selecting the entries in the dictionary. This problem can be easily mapped to a variant of the clique partitioning problem from graph theory [5]. We next describe the clique partitioning problem and then show how the problem of determining dictionary entries can be mapped to this problem. We then present a heuristic algorithm for generating the dictionary entries. The proposed algorithm presents a graph-theoretic view of the procedure presented in [15].

An undirected graph G consists of a set of vertices V and a set of edges E, where each edge connects an unordered pair of vertices. Given an undirected graph $G = (V, E)$, a *clique* of the graph is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E [5]. Given a positive integer K, the clique partitioning problem refers to the partitioning of V into k cliques, where $k \leq K$. The clique partitioning problem is $\mathcal{NP}$-hard [8][1], hence heuristic approaches must be used to solve it in reasonable time for large problem instances.

Recall that in dictionary-based data compression, we obtain nl m-bit words after placing the test set in a multiple scan chain format. Two words $u_1u_2 \ldots u_m$ and $v_1v_2 \ldots v_m$ are defined to be *compatible* to each other if for any position i, u_i and v_i are either equal to each other or at least one of them is a don't-care bit. We construct an undirected graph G to reflect the compatible relationships between the words as follows. First, a vertex is added to the graph for each word. Then we examine each pair of words. If they are mutually compatible, an edge is added between the corresponding pair of vertices. A clique in G refers to a group of test data words that can be mapped to the same dictionary entry. If the dictionary can have at most $|D|$ entries and the total number of words is nl, the goal of the compression procedure is to find the largest subset of G that can be partitioned into $|D|$ cliques; the remaining vertices in G denote test data words that are not compressed. This problem can easily be shown to be $\mathcal{NP}$-hard by contradiction. If the compression can be optimally solved in polynomial time then it provide a yes/no answer to the decision version of the clique partitioning problem in polynomial time. We therefore use the following

[1] The decision version of the clique partitioning problem is $\mathcal{NP}$-complete.

TABLE I. AN EXAMPLE OF TEST DATA FOR MULTIPLE SCAN CHAINS.

Scan chain index	Word index															
	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	0	1	1	1	1	0	0	0	X	X	0	X	0	1
2	X	0	1	1	0	0	1	X	X	X	1	X	X	0	1	0
3	X	X	X	X	0	X	0	1	0	0	1	1	0	X	X	X
4	X	0	X	0	X	X	0	X	0	0	0	0	0	X	0	1
5	0	0	0	0	X	0	X	0	X	X	X	0	X	1	0	X
6	0	X	1	0	1	0	X	X	1	X	0	0	X	0	X	X
7	1	0	1	X	X	X	X	1	1	0	X	1	0	0	1	0
8	1	X	0	X	0	1	X	1	0	X	X	X	X	X	X	1

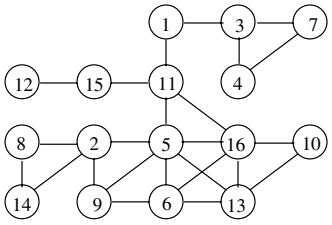

Fig. 3. The graph G for the example of Table I.

heuristic procedure.

1) Copy the graph G to a temporary data structure G'.
2) Find the vertex v with the maximum degree in G'.
3) Establish a subgraph that consists of all the vertices connected to v. Copy this subgraph to G' and add v to a set C.
4) If G' is not empty, go to Step 2. Otherwise, a clique C has been formed consisting all the vertex found in Step 2.
5) Remove the vertices in the clique C from G and copy $G - C$ to G'. Go to Step 2 and repeat until $|D|$ cliques are found.

The complexity of this procedure is $O(N^3)$, where $N = nl$ is the number of vertices in the graph. Table I shows an example of test data formatted for multiple scan chains. The number of scan chains m is 8 in this example. There are total of 16 words, each of which has 8 bits. Figure 3 shows the corresponding graph G for the test data. Let us assume that a dictionary of size four is to be formed, i.e. $|D| = 4$. Using the greedy algorithm described above, we obtain four cliques: {5, 6, 13, 16}, {2, 8, 14}, {3, 4, 7} and {1, 11}. (Here we use the word indices of Table I to represent the vertices.) After finding the cliques, we obtain the corresponding dictionary entry for each clique by merging the words in this clique. In this example, the four dictionary entries are {11100011, 01000110, 0000100X, 10X10001}. Three bits are then needed to encode the words in the cliques; an additional 1 bit is needed for the prefix, and 2 bits are required for the dictionary index. For words that are not in any clique, a total of 9 bits each must be transferred from the ATE. Since there are 12 words that can be generated from the dictionary, the size of the compressed data is $3\times12+9\times4 = 72$ bits, which corresponds to a compression of 43.75%. Moreover, the dictionary entries still contain some don't-care bits, which can reduce the hardware for the decoder, as explained next.

The clique partitioning procedure introduces a certain degree of randomness in the way the don't-care bits in T_D are filled; the resulting "random fill" can be expected to increase the fortuitous detection of non-modeled faults. This is in contrast to coding methods such as [3, 4] in which the don't-cares are all mapped to 0s.

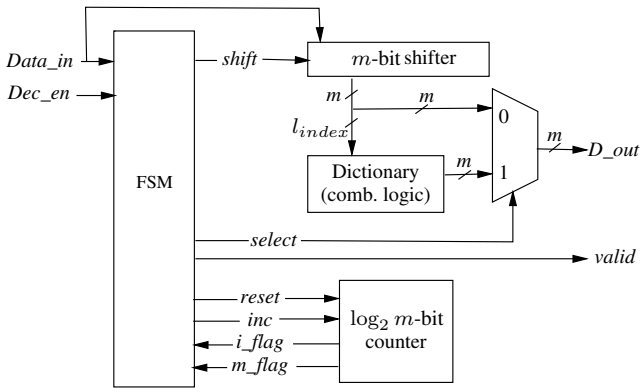

Fig. 4. Decompression Architecture I.

III. DECOMPRESSION ARCHITECTURE

In this section, we describe the decompression architecture for the dictionary-based test data compression method. Two architectures for the on-chip decoder are proposed; they are referred as Architecture I and Architecture II, respectively. Architecture I does not require any modification to the scan chains of the core under test. Architecture II is intended for cores with flexible scan chains since a slight modification to the scan chains is needed in this case. The latter requires less hardware than Architecture I.

Figure 4 illustrates the first decompression architecture. The decoder consists of a finite-state machine (FSM), an m-bit shifter, a $\log_2 m$-bit counter, a selector and the dictionary. The m-bit shifter takes data from $Data\_in$ when the signal $shift$ equals 1. All the m output bits pass to the selector while the dictionary only gets the higher order l_{index} bits of the output as its index. Here we use a combinational logic circuit to implement the dictionary. It outputs the m-bit word corresponding to the current value of the index.

The $\log_2 m$-bit counter is used to indicate whether the shift-in of a codeword has finished. It operates as follows.

- The signal $reset$ resets the counter to 0.
- If $inc = 1$, the counter is incremented.
- When the value of the counter reaches l_{index}, the output i_flag equals 1.
- When the value of the counter reaches m, the output m_flag equals 1.

The FSM has only three states. It is enabled when the signal Dec_en is active. It starts by checking the first bit of the data shifted in from $Data\_in$. If this bit is 0, which implies that the next m bits directly constitute a word, the FSM shifts this word into the m-bit shifter, which then feeds this word into the scan chains during the clock cycle in which the decoder checks the first bit of the next codeword. On the other hand, if the first bit is 1, the FSM shifts in the next l_{index} bits and gets the decoded word from the dictionary. Only 19 gates and 2 flip-flops are required to implement the FSM using Synopsys

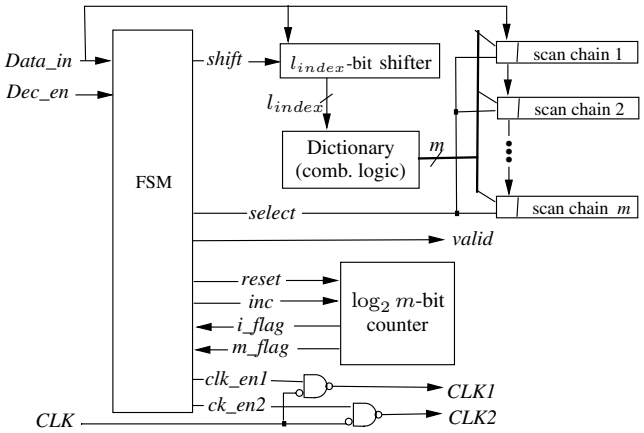

Fig. 5. Decompression Architecture II.

Design Compiler.

In Architecture I, an m-bit shifter is needed for the on-chip decoder. If the number of scan chains is large, then the hardware overhead resulting from this shifter is also large. If a small modification to the scan chain is permitted, e.g., in the case of flexible scan chains, this m-bit shifter can be replaced by a small l_{index}-bit shifter. Figure 5 illustrates this decompression architecture. Here, the scan chains are slightly modified such that the first cell in each scan chain can receive the input test data either from the dictionary or from the first bit of the previous scan chain. The signal $select$ is used to select the source of the input test data for the first cell of the scan chains. The modification to a scan chain is therefore limited to its first scan cell. This can be implemented in a non-intrusive fashion by adding a multiplexer at the scan input pin. During decompression, the controlled clock signal $CLK1$ is used to drive the first cells of the scan chains, and $CLK2$ drives the remaining flip-flops in the scan chains and the MISR. The operation of the FSM is now slightly different from that of Architecture I for the codewords that consist of a 0 followed by a word. If the first bit of the codeword is 0, $CLK1$ is enabled for the next m cycles and $CLK2$ is enabled only for the next cycle. Thus in the next cycle, the response in the scan chains are shifted one bit towards the MISR and one bit data is shifted into the first cell of the first scan chain. During the subsequent $m-1$ cycles, the remaining $m-1$ bits data are shifted into the first cells of the scan chains and other cells in the scan chains are frozen by disabling the clock signal $CLK2$. The FSM for Architecture II has only 3 states, and 21 gates and 2 flip-flops are required to implement it using Synopsys Design Compiler.

We can further reduce hardware overhead by using a single dictionary for several cores. We concatenate the test sets for the cores and view the concatenated test data as a single entity during compression. The cores that use the same dictionary are tested serially during the test session.

In the above decompression architectures, it is assumed that only one ATE channel is used to transfer the compressed data. The extension to multiple channels is straightforward. For example, if the length of the dictionary index is 7, then a codeword consists of 8 bits (1-bit prefix and 7-bit index). For such a configuration, we can use either 2, 4 or 8 ATE channels and an on-chip register of appropriate size without wasting ATE channel bandwidth.

IV. Experimental Results

In this section, we first apply the dictionary-based test data compression method to the test cubes for the seven largest ISCAS-89 circuits. These test cubes were obtained from the Mintest ATPG program [10]. We refer to the uncompressed test set as T_D and the compressed test set as T_E. For each test set, we determine a dictionary that contains 128 entries, which provides good tradeoff between compression and hardware overhead. Thus the length of the dictionary index is 7 bits. Table II shows the compression results for a varying number of scan chains. The second and third columns list the number of test vectors in each test set and the number of scan cells in each circuit, respectively. The minimum size of T_E for each circuit is shown in boldface. The last column of the table shows the hardware overhead for the dictionary corresponding to the value of m for which the minimum size of T_E is obtained. For instance, the choice of $m = 64$ minimizes the size of T_E for circuit s5378, and 656 gates are required to implement the corresponding dictionary, which has 7 inputs and 64 outputs. Since no additional handshaking is required between the ATE and the SOC, the testing time (in ATE clock cycles) is simply equal to the size of T_E.

Table III presents a comparison of the compression results with EFDR coding [7], variable-length input Huffman coding (VIHC) [9], RESPIN++ [21], XOR network encoding [2], and test data mutation encoding [18]. The size of T_E listed for the dictionary-based method is the minimum size shown in boldface in Table II. We find that the dictionary-based method outperforms EFDR coding, VIHC coding and RESPIN++ for all circuits. The result for RESPIN++ in [21] are based on a preprocessing step of 400 pseudorandom patterns. Despite the fact that the deterministic patterns in [21] targeted a smaller number of faults than in this work, we obtain higher compression than [21]. Compared to XOR network encoding, the compression is higher for four out of five circuits. There are two sets of compression results for test data mutation encoding in [18]. They are obtained by encoding Mintest test sets (1) and Atalanta-generated test sets (2), respectively. The last column lists the compression results obtained by combining test data mutation encoding and XOR network encoding, as well as additional structural information for test generation and compaction. Compared to these three sets of results, the dictionary-based method achieves higher compression for at least three out of five circuits for each case.

In [19], all distinct output m-bit words for m scan chains are included in the dictionary. This leads to a prohibitively large dictionary, especially for large values of m. The statistical coding method of [15] is based on dictionaries with variable-length indices. For comparison, we implemented this method and found that it provides higher compression. However, the finite-state machine decoder of [15] requires 2–3 times more

TABLE II. COMPRESSION RESULTS WITH VARYING NUMBER OF SCAN CHAINS.

Circuit	No. of test vectors	No. of scan cells	Size of T_D (bits)	Size of T_E for varying no. of scan chains (bits)						No. of gates for the dictionary
				$m = 16$	$m = 32$	$m = 48$	$m = 64$	$m = 128$	$m = 200$	
s5378	111	214	23754	12999	7791	6572	**6345**	8794	12970	656
s9234	159	247	39273	21189	13551	13659	12783	**11498**	16826	951
s13207	236	700	165200	83882	43936	31231	24074	13990	**8517**	1118
s15850	126	611	76986	40491	24960	19705	18573	**13873**	14840	1093
s35932	16	1763	28208	17214	8818	11583	7403	3123	**1400**	640
s38417	99	1664	164736	92304	84009	**62939**	94350	104192	114243	581
s38584	136	1464	199104	107962	71148	63740	58027	58189	**53287**	1469

TABLE III. COMPARISON OF COMPRESSION RESULTS.

Circuit	Size of T_D (bits)	Size of T_E (bits)							
		Dictionary based	EFDR [7]	VIHC [9]	RESPIN++ [21]	XOR network [2]	Mutation encoding [18]		Mutation encoding + XOR network [18]
							(1)	(2)	
s5378	23754	6345	11419	11516	17332	N/A	N/A	N/A	N/A
s9234	39273	11498	21250	17736	17198	N/A	N/A	N/A	N/A
s13207	165200	8517	29992	27737	26004	25344	74423	16913	15783
s15850	76986	13873	24643	30271	32226	22784	26021	14676	10798
s35932	28208	1400	5554	9458	N/A	7128	7222	11298	3972
s38417	164736	62939	64962	74938	89132	89856	45003	55848	42264
s38584	199104	53287	73853	85674	63232	38976	73464	47886	22636

TABLE IV. COMPRESSION RESULTS ON USING A SINGLE DICTIONARY FOR TWO CIRCUITS.

Circuit pair	Size of T_D (bits)	No. of scan chains (m)	No. of gates for the dictionary	Size of T_E (bits)								
				Dictionary based		EFDR [7]	VIHC [9]	RESPIN++ [21]	XOR network [2]	Mutation encoding [18]		Mutation encoding + XOR network [18]
				Joint	Separate					(1)	(2)	
{s5378, s9234}	63027	32	525	24642	21342	32669	29252	34530	N/A	N/A	N/A	N/A
{s13207, s15850}	242186	128	1175	40810	27863	54635	58008	58230	48128	100444	31589	26581
{s38417, s38584}	363840	48	689	150090	126679	138815	160612	152364	128832	118467	103734	64900

TABLE V. COMPRESSION RESULTS FOR TEST DATA FROM IBM.

Circuit	Size of T_D (bits)	No. of scan chains (m)	Size of T_E (bits)	Percentage compression	No. of gates for the dictionary	Size of T_E (bits) for [15], and compression	No. of gates for the FSM in [15]
CKT1	11613472	400	232824	98.00	1541	122407 (98.95%)	6256
CKT2	4124288	200	172519	95.82	1917	136118 (96.70%)	4283

chip area than the proposed fixed-length dictionary.

Note that even smaller test data volume has been reported recently for these circuits using a combination of packetization and data compression codes [23]. These numbers were however obtained after a preprocessing step involving pseudorandom patterns, therefore a direct comparison can be misleading. Test data compression of up to 100X has also been reported recently for industrial designs using commercial tools [17]. This approach however relies on the use of test generation together with test data compression—less compression might be expected for precomputed tests. A direct comparison with [17] is also difficult due to the proprietary nature of the underlying compression method and the industrial designs. In our work, we neither use a preprocessing step involving pseudorandom patterns nor do we interleave test generation with test data compression. These additional steps can also be used to enhance the effectiveness of dictionary-based compression.

Table IV shows compression results when a single dictionary is used for pairs of ISCAS-89 circuits. We consider pairs of circuits that have nearly the same number of scan cells. The third column in the table lists the number of scan chains with which the dictionary-based method is used. The test data volume for the competing methods is calculated by summing the test data volumes for the individual circuits. We see that even though a single dictionary is used for the two circuits, the compression is greater than that for VIHC coding and RESPIN++ in all cases, also better than that for EFDR coding in two out of three cases, and it is higher than XOR-network encoding and test data mutation encoding in one out of two cases.

In order to evaluate the compression efficiency of the dictionary-based method for large test sets, we applied the method to two real test sets from industry. The test set for the first circuit (CKT1) from IBM consists of 32 statically-compacted scan vectors (a total of 362921 bits of test data per vector). This microprocessor design consists of 3.6 million gates and 726000 latches. The test set for a second microprocessor circuit (CKT2) from IBM consists of a set of 4 scan vectors (a total of 1031072 bits of test data per vector); this design contains 1.2 million gates and 32200 latches. Table V lists the information about the test sets and the compression results for these two industrial circuits. In the absence of information about the number of scan chains for CKT1 and CKT2, we assumed a total of 400 scan chains for CKT1 and 200 scan chains for CKT2. The sixth column of Table V lists the number of gates required to implement the corresponding dictionaries with Synopsys Design Compiler. For CKT1 and CKT2, we ob-

tain 50X and 25X compression, respectively, and the hardware overhead due to the dictionaries is negligible. We applied the technique of [15] to these test sets for block sizes of 400 and 200, respectively. As shown in Table V, the compression is greater, but the hardware overhead is substantially higher for the coding method of [15].

Note that, as in [9, 15, 19], the dictionaries presented are specific to the circuit under test. It is often desirable however to use a circuit-independent dictionary, which as in [3, 4], makes the decompression logic independent of the precomputed test set. This is especially useful in cases of circuit redesign or test set modifications. In the proposed approach, the decompression logic is test-independent if the dictionary is transferred from the ATE to an embedded RAM at the start of a test session. If this is not the case, and the dictionary is implemented as a custom combinational logic, we can still use the same dictionary for several circuits, albeit with a potential decrease in the amount of compression. The negative impact on compression can however be minimized if a dictionary for a small circuit (CKT A) is used for a large circuit (CKT B). The relatively smaller number of test patterns for CKT A leaves many don't-cares in the dictionary, which in turn can be used to efficiently match the test patterns for CKT B. For example, we compressed T_D for s13207 using the dictionary for s5378 and found that T_E in this case contains 23571 bits, which is still less than the test data volume for several competing methods. In another experiment aimed at evaluating the effect of test set modifications, we first generated a dictionary for s38584 with $m = 32$ and then randomly altered 10% of the test vectors in T_D. The resulting compression obtained with the original dictionary is only 6% less than before. We are currently investigating this aspect of dictionary-based compression in more detail.

V. Conclusion

We have shown how dictionary-based test data compression can be used to reduce test data volume and testing time for SOCs. The proposed method delivers compressed patterns from the tester to the chip and drives a large number of internal scan chains using only a single ATE channel. Hence the dictionary-based compression technique is especially suitable for reduced-pin count testing, multi-site testing, as well as a low-cost DFT test environment. In contrast to techniques based on data compression codes, this approach does not rely on additional synchronization between the SOC and the ATE. Experimental results for the ISCAS-89 and representative test data from IBM show that the dictionary can be implemented with a small amount of hardware and the test data volume for the proposed method is less than that for a number of recently-proposed test data compression techniques. We are currently carrying out a more detailed comparison between the proposed method and compression based on dictionaries with variable-length indices.

Acknowledgment

We thank Prof. Nur Touba of University of Texas at Austin for valuable comments on a draft version of this paper.

References

[1] C. Barnhart et al., "OPMISR: the foundation for compressed ATPG vectors", *Proc. Int. Test Conf.*, pp. 748–757, 2001.

[2] I. Bayraktaroglu and A. Orailoglu, "Test volume and application time reduction through scan chain concealment", *Proc. ACM/IEEE Design Automation Conf.*, pp. 151–155, 2001.

[3] A. Chandra and K. Chakrabarty, "Frequency-directed run-length (FDR) codes with application to system-on-a-chip test data compression," *Proc. VLSI Test Symp.*, pp. 42-47, 2001.

[4] A. Chandra and K. Chakrabarty, "System-on-a-chip test data compression and decompression architectures based on Golomb codes", *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 355–368, March 2001.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT press, Cambridge, London, England, 2001.

[6] A. El-Maleh, S. al Zahir, and E. Khan, "A geometric-primitives-based compression scheme for testing systems-on-chip", *Proc. VLSI Test Symp.*, pp. 54–59, 2001.

[7] A. El-Maleh and R. Al-Abaji, "Extended Frequency-Directed Run-Length Codes with Improved Application to System-on-a-Chip Test Data Compression", *Proc. Int. Conf. Electronics, Circuits and Systems*, pp. 449–452, 2002.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[9] P. T. Gonciari, B. Al-Hashimi and N. Nicolici, "Improving compression ratio, area overhead, and test application time for system-on-a-chip test data compression/decompression", *Proc. Design, Automation and Test in Europe Conf.*, pp. 604–611, 2002.

[10] I. Hamzaoglu and J. H. Patel, "Test set compaction algorithms for combinational circuits", *Proc. Int. Conf. CAD*, pp. 283-289, 1998.

[11] S. Hellebrand, H.-G. Liang and H.-J. Wunderlich, "A mixed-mode BIST scheme based on reseeding of folding counters", *Proc. Int. Test Conf.*, pp. 778–784, 2000.

[12] F. F. Hsu, K. M. Butler and J. H. Patel, "A case study on the implementation of Illinois scan architecture", *Proc. Int. Test Conf.*, pp. 538–547, 2001.

[13] V. Iyengar, K. Chakrabarty and B. T. Murray, "Deterministic built-in pattern generation for sequential circuits," *JETTA*, vol. 15, pp. 97-115, 1999

[14] A. Jas and N. A. Touba, "Test vector decompression via cyclical scan chains and its application to testing core-based design", *Proc. Int. Test Conf.*, pp. 458–464, 1998.

[15] A. Jas, J. Ghosh-Dastidar and N. A. Touba, "Scan vector compression/decompression using statistical coding," *Proc. VLSI Test Symp.*, pp. 114-120, 1999.

[16] B. Koenemann et al., "A SmartBIST variant with guaranteed encoding," *Proc. Asian Test Symp.*, pp. 325–330, 2001.

[17] J. Rajski et al., "Embedded deterministic test for low-cost manufacturing test," *Proc. Int. Test Conf.*, pp. 301–310, 2002.

[18] S. Reda and A. Orailoglu, "Reducing test application time through test data mutation encoding", *Proc. Design, Automation and Test in Europe Conf.*, pp. 387-393, 2002.

[19] S. M. Reddy, K. Miyase, S. Kajihara, and I. Pomeranz, "On test data volume reduction for multiple scan chain design", *Proc. VLSI Test Symp.*, pp. 103–108, 2002.

[20] D. Salomon, *Data Compression: The Complete Reference*, Springer-Verlag New York, Inc., New York, NY, 2000.

[21] L. Schafer, R. Dorsch, and H.-J. Wunderlich, "RESPIN++ – Deterministic Embedded Test", *Proc. European Test Workshop*, pp. 37-44, 2002.

[22] N. A. Touba and E. J. McCluskey, "Altering a pseudo-random bit sequence for scan based BIST", *Proc. Int. Test Conf.*, pp. 167–175, 1996.

[23] E. H. Volkerink, A. Khoche and S. Mitra, "Packet-based input test data compression techniques", *Proc. Int. Test Conf.*, pp. 154–163, 2002.

[24] H. Vranken, T. Waayers, H. Fleury and D. Lelouvier, "Enhanced reduced pin-count test for full-scan designs", *Proc. Int. Test Conf.*, pp. 738–747, 2001.

[25] F. G. Wolff and C. Papachristou, "Multiscan-based test compression and hardware decompression using LZ77", *Proc. Int. Test Conf.*, pp. 331–339, 2002.

[26] H.-J. Wunderlich and G. Kiefer, "Bit-flipping BIST", *Proc. Int. Conf. Computer-Aided Design*, pp. 337-343, 1996.